

# TD 11: XSLT

## Rappels / Outils

- Chapitre de cours: [XSLT](#)
- Appliquer une stylesheet XSLT à un fichier XML: `xsltproc stylesheet.xml fichier.xml`

## Reprise de l'exercice 1 du [TD 10](#): À finir en une heure maximum!

On va travailler avec un [exemple de fichier XML](#) de bibliographie, dont un **petit extrait** est montré ici:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bibtex:file xmlns:bibtex="http://bibtexml.sf.net/">

<bibtex:entry id="berkman">
  <bibtex:book>
    <bibtex:author>Berkman, R. I.</bibtex:author>
    <bibtex:title>
      <bibtex:title>Find It Fast</bibtex:title>
      <bibtex:subtitle>How to Uncover Expert Information
        on Any Subject</bibtex:subtitle>
    </bibtex:title>
    <bibtex:publisher>HarperPerennial</bibtex:publisher>
    <bibtex:year>1994</bibtex:year>
    <bibtex:address>New York</bibtex:address>
  </bibtex:book>
</bibtex:entry>

<bibtex:entry id="moir">
  <bibtex:book>
    <bibtex:author>
      <bibtex:person>Moir, A.</bibtex:person>
      <bibtex:person>Jessel, D.</bibtex:person>
    </bibtex:author>
    <bibtex:title>
      <bibtex:title>Brain Sex</bibtex:title>
      <bibtex:subtitle>The Real Difference
        Between Men and Women</bibtex:subtitle>
    </bibtex:title>
    <bibtex:publisher>Mandarin</bibtex:publisher>
    <bibtex:year>1991</bibtex:year>
    <bibtex:address>London</bibtex:address>
  </bibtex:book>
</bibtex:entry>

</bibtex:file>
```

1. Écrire une feuille de style XSL qui produit un document HTML contenant une table à 2 colonnes: titre et auteur(s) de chaque entrée.

Précisions / Aides:

- Utilisez le bon type de `xsl:output`!
- Rappel: Pour matcher des éléments de la namespace `bibtex`, cette namespace doit être déclarée (correctement!) dans votre stylesheet.
- Attention: certaines entrées n'ont pas de titre! Laissez la case correspondante vide.
- Dans un premier temps, ne faites pas attention au formatage des **auteurs**, et mettez-les en vrac.

Exemple *indicatif* de HTML souhaité en sortie (source à gauche; HTML à droite):

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>TD 10 - Exo 1</title></head>
<body>
  <table border=1 cellpadding=2 cellspacing=0>
    <tr>
      <th>Titre</th>
```

```
<th>Auteurs</th>
</tr>
<tr>
  <td>L'essentiel de XML</td>
  <td>Olivier Carton</td>
</tr>
<tr>
  <td>War and Peace</td>
  <td>Leon Tolstoï</td>
</tr>
</table>
</body>
</html>
```

Titre	Auteur
L'essentiel de XML	Olivier Carton
War and Peace	Leon Tolstoï

Voir un [aperçu du résultat demandé](#). Corrigés: [sortie HTML](#), [XSL](#)

2. Essayez-le avec le fichier exemple!

## Exercice 2: Calcul et `xsl:call-template`

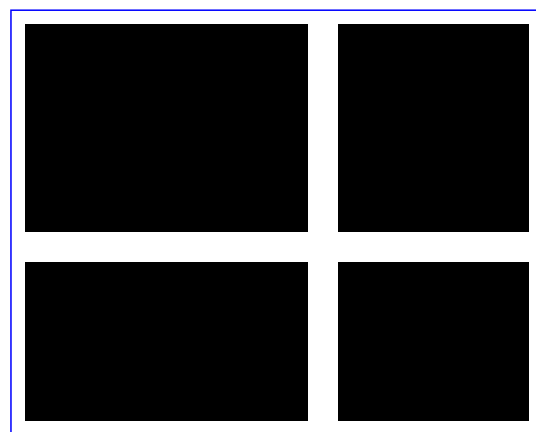
1. Que fait `points.xslt` lorsqu'il est appliqué à `points.xml` ?
2. Modifiez `points.xslt` pour qu'il affiche deux nouvelles lignes:
  - l'une contenant la moyenne du total hebdomadaire de points, sur l'ensemble des semaines.
  - l'autre contenant la moyenne des points par jour, sur l'ensemble des jours.
3. Refaites la question précédente, **sans utiliser `count()`** !
4. Ajoutez l'affichage du nombre maximal de points obtenu en un jour.
5. Ajoutez l'affichage de la variance du nombre de points par jour.
6. Ecrivez maintenant votre propre fichier XSLT, qui transforme `points.xml` en `info.xml`, où les balises `<jour>` sont recopiées, de manière à ce que le(s) jour(s) avec le maximum de points aient un attribut `remarque="max"`, et que les autres jours aient un attribut `remarque="plus"` ou `remarque="moins"` selon leur position par rapport à la moyenne.

## Exercice 3: SVG

On travaille sur [un fichier SVG](#) (à gauche, son contenu; à droite son rendu; tel que vous le voyez en ouvrant le fichier `.svg` dans un navigateur Web):

```
<?xml version="1.0" standalone="no"?>
<svg width="700" height="560" version="1.1"
  xmlns="http://www.w3.org/2000/svg">
  <desc>Four separate rectangles
</desc>
<rect x="20" y="20" width="370" height="272"/>
<rect x="430" y="20" width="250" height="272"/>
<rect x="20" y="332" width="370" height="208"/>
<rect x="430" y="332" width="250" height="208"/>

<!-- Show outline of canvas using 'rect' element -->
<rect x="1" y="1" width="698" height="558"
  fill="none" stroke="blue" stroke-width="2" />
</svg>
```



1. Écrire une stylesheet XSLT qui, appliquée au fichier `3.svg`, donne le nombre de rectangles.  
Aides:
  - Quand le document source déclare une namespace par défaut (c'est le cas ici!); il est impératif de re-déclarer cette namespace dans la feuille de style XSLT, en lui **donnant un préfixe** non vide. Vos expressions XPath devront ensuite utiliser ce préfixe pour référencer des éléments. [Explication sur stackoverflow](#)
  - Utilisez `<xsl:text>&#xa;</xsl:text>` pour ajouter un saut de ligne.

Exemple de sortie souhaitée: `Nombre de rectangles: 5`

[Corrigé](#)

2. Modifiez votre XSLT pour afficher également les aires des rectangles. On supposera que toutes les unités du SVG sont en centimètres.

Exemple de sortie souhaitée:

```
Nombre de rectangles: 5
Aires des rectangles:
100640 pixels
68000 pixels
76960 pixels
52000 pixels
389484 pixels
```

[Corrigé](#)

3. Modifiez votre XSLT pour qu'il ne prenne en compte que les rectangles **remplis** (regarder l'attribut `fill`), et qu'il calcule les aires en **pourcentage** de la surface totale.

Exemple de sortie souhaitée:

Nombre de rectangles: 4  
Aires des rectangles:  
25.6734693877551 %  
17.3469387755102 %  
19.6326530612245 %  
13.265306122449 %

### Corrigé

4. Créez un XSLT qui transforme un SVG source en le même SVG, réduit de 50%.

Aides:

- o N'oubliez pas d'ajuster `xsl:output`
- o Si on éditait directement le fichier SVG, la solution la plus simple serait d'insérer un élément `<g>` juste en dessous de l'élément racine `<svg>` avec un attribut `transform` adéquat (cf [SVG reference](#))
- o Le XSLT peut utiliser `<xsl:copy>` pour recopier l'élément courant (sans ses attributs ni contenu), et `<xsl:copy-of>` pour recopier des parties d'un élément, telles que ses attributs, et/ou son contenu.

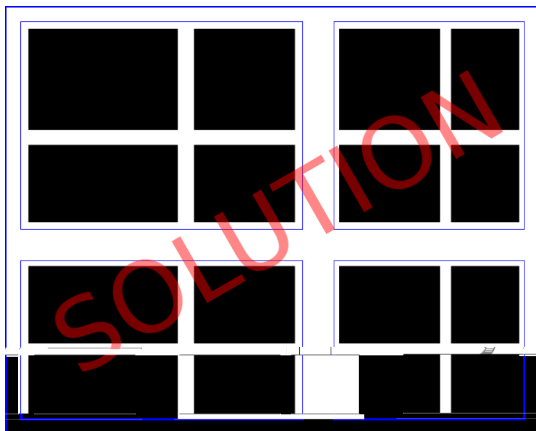
Appliquez ce XSLT au fichier 3.svg: `xsltproc votre.xsl 3.svg > 3.4.svg`, et ouvrez la sortie 3.4.svg dans votre navigateur.

Réparez votre XSLT jusqu'à ce que ça marche!

### Corrigé

5. (\*) Créer un XSLT qui transforme 3.svg en remplaçant chaque rectangle **rempli** par une copie *réduite* du document SVG originel (correspondant à la taille du rectangle remplacé).

Ça devrait ressembler a l'image ci-dessous:



### Corrigé

6. Modifiez, si besoin, le XSLT pour qu'il puisse s'appliquer récursivement.

Appliquez-le 3 fois:

```
xsltproc votre.xsl 3.svg > 3.6.svg  
xsltproc votre.xsl 3.6.svg > 3.66.svg  
xsltproc votre.xsl 3.66.svg > 3.666.svg
```

Et vérifiez qu'il ressemble à ça:



[Corrigé](#)

7. (\*\*) En s'aidant du graphique ci-contre (cliquer pour l'agrandir), créer un XSLT qui permet, en l'appliquant récursivement, de transformer toute image SVG de proportions 200x450 en [fougère de Barnsley](#).

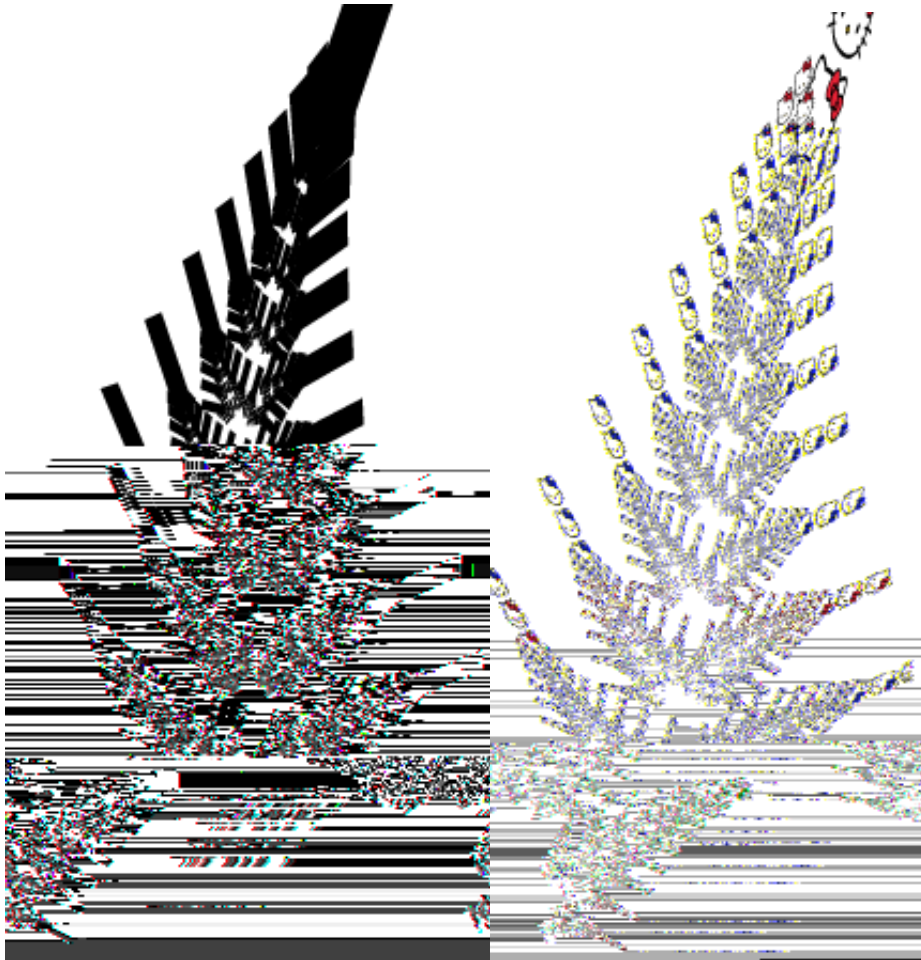
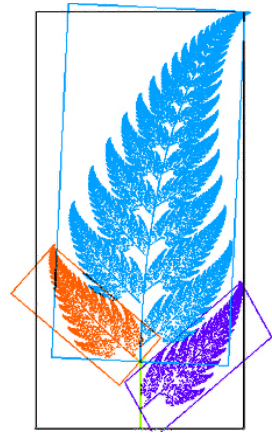
Pour ajuster vos transformations, utilisez le fichier [3.7.svg](#): `xsltproc votre.xsl 3.7.svg > test.svg`.

Une fois ajusté; appliquez-le 10 fois de suite sur [3.7.0.svg](#) à l'aide de cette commande (copiez-collez tout):

```
XSL=votre.xsl
for i in {0..10}; do
  echo "Iteration #${i}"
  (( j=i+1 ))
  xsltproc $XSL 3.7.${i}.svg > 3.7.${j}.svg
  j=${i}
done
```

Vous pouvez également essayer avec d'autres images 200x450, comme [Hello Kitty](#) (en le sauvegardant sous `3.7.0.svg`, le script marchera sans modifications).

Ca devrait ressembler à ça:



[Corrigé](#)